PATENT COOPERATION TREATY

From the
INTERNATIONAL SEARCHING AUTHORITY

To: see form PCT/ISA/220	PCT WRITTEN OPINION OF THE INTERNATIONAL SEARCHING AUTHORITY (PCT Rule 43*bis*.1)
	Date of mailing *(day/month/year)* see form PCT/ISA/210 (second sheet)
Applicant's or agent's file reference see form PCT/ISA/220	**FOR FURTHER ACTION** See paragraph 2 below

International application No. PCT/GB2004/004551	International filing date *(day/month/year)* 28.10.2004	Priority date *(day/month/year)* 28.10.2003
International Patent Classification (IPC) or both national classification and IPC G06F9/445		
Applicant SYMBIAN SOFTWARE LIMITED		

1. This opinion contains indications relating to the following items:

☒	Box No. I	Basis of the opinion
☒	Box No. II	Priority
☐	Box No. III	Non-establishment of opinion with regard to novelty, inventive step and industrial applicability
☐	Box No. IV	Lack of unity of invention
☒	Box No. V	Reasoned statement under Rule 43*bis*.1(a)(i) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement
☐	Box No. VI	Certain documents cited
☐	Box No. VII	Certain defects in the international application
☐	Box No. VIII	Certain observations on the international application

2. **FURTHER ACTION**

If a demand for international preliminary examination is made, this opinion will usually be considered to be a written opinion of the International Preliminary Examining Authority ("IPEA"). However, this does not apply where the applicant chooses an Authority other than this one to be the IPEA and the chosen IPEA has notifed the International Bureau under Rule 66.1*bis*(b) that written opinions of this International Searching Authority will not be so considered.

If this opinion is, as provided above, considered to be a written opinion of the IPEA, the applicant is invited to submit to the IPEA a written reply together, where appropriate, with amendments, before the expiration of three months from the date of mailing of Form PCT/ISA/220 or before the expiration of 22 months from the priority date, whichever expires later.

For further options, see Form PCT/ISA/220.

3. For further details, see notes to Form PCT/ISA/220.

Name and mailing address of the ISA: European Patent Office D-80298 Munich Tel. +49 89 2399 - 0 Tx: 523656 epmu d Fax: +49 89 2399 - 4465	Authorized Officer Kalejs, E Telephone No. +49 89 2399-6919

Box No. I Basis of the opinion

1. With regard to the **language**, this opinion has been established on the basis of the international application in the language in which it was filed, unless otherwise indicated under this item.

   ☐ This opinion has been established on the basis of a translation from the original language into the following language , which is the language of a translation furnished for the purposes of international search (under Rules 12.3 and 23.1(b)).

2. With regard to any **nucleotide and/or amino acid sequence** disclosed in the international application and necessary to the claimed invention, this opinion has been established on the basis of:

   a. type of material:

      ☐ a sequence listing

      ☐ table(s) related to the sequence listing

   b. format of material:

      ☐ in written format

      ☐ in computer readable form

   c. time of filing/furnishing:

      ☐ contained in the international application as filed.

      ☐ filed together with the international application in computer readable form.

      ☐ furnished subsequently to this Authority for the purposes of search.

3. ☐ In addition, in the case that more than one version or copy of a sequence listing and/or table relating thereto has been filed or furnished, the required statements that the information in the subsequent or additional copies is identical to that in the application as filed or does not go beyond the application as filed, as appropriate, were furnished.

4. Additional comments:

Box No. II Priority

1. ☒ The following document has not been furnished:

   ☒ copy of the earlier application whose priority has been claimed (Rule 43*bis*.1 and 66.7(a)).

   ☐ translation of the earlier application whose priority has been claimed (Rule 43*bis*.1 and 66.7(b)).

   Consequently it has not been possible to consider the validity of the priority claim. This opinion has nevertheless been established on the assumption that the relevant date is the claimed priority date.

2. ☐ This opinion has been established as if no priority had been claimed due to the fact that the priority claim has been found invalid (Rules 43*bis*.1 and 64.1). Thus for the purposes of this opinion, the international filing date indicated above is considered to be the relevant date.

3. ☐ It has not been possible to consider the validity of the priority claim because a copy of the priority document was not available to the ISA at the time that the search was conducted (Rule 17.1). This opinion has nevertheless been established on the assumption that the relevant date is the claimed priority date.

4. Additional observations, if necessary:

Box No. V Reasoned statement under Rule 43*bis*.1(a)(i) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement

1. Statement

Novelty (N)	Yes:	Claims	
	No:	Claims	1-10
Inventive step (IS)	Yes:	Claims	
	No:	Claims	1-10
Industrial applicability (IA)	Yes:	Claims	1-10
	No:	Claims	

2. Citations and explanations

   see separate sheet

Re Item V

Reasoned statement with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement

1. The following document is referred to in this communication; the numbering will be adhered to in the rest of the procedure:

   D1: "FORWARDER DYNAMIC LINK LIBRARIES AS A METHOD FOR SERVICING SOFTWARE" IBM TECHNICAL DISCLOSURE BULLETIN, IBM CORP. NEW YORK, US, vol. 38, no. 11, 1 November 1995 (1995-11-01), pages 407-408, XP000547408 ISSN: 0018-8689

2. Lack of novelty, Article 33(2) PCT

3.1 The present application does not meet the requirements of Article 33(2) PCT, because the subject-matter of claim 1 is not new in the sense of Article 33(2) PCT.

The document D1 discloses (the references in parentheses applying to this document):

A method of providing a dynamic link library for linking between functions and an executable program in a computing device, the method comprising providing the dynamic link library as:
- a first part for linking the executable program to one or more first functions (page 407, figure 2. The primary DLL 1 and primary DLL 2 are interpreted as the first part); and
- an extension part for causing the executable program to link one or more further functions, additional to the one or more first functions, via the extension part (page 407, figure 2. The Forwarded DLLs 1-3 are interpreted as the extension part).

3.2 Dependent claims 2-8 do not contain any features which, in combination with the features of any claim to which they refer, meet the requirements of the PCT in respect of novelty and/or inventive step, the reasons being as follows:

The features disclosed in claims 2-8 are well-known per se and represent, in fact, merely a few of several straightforward possibilities from which the skilled person would select, in accordance with circumstances, without the exercise of inventive skill, in order to solve the problem posed.

3.3 Claims 9 and 10 claim the subject-matter of claims 1-8 in the form of a device and computer software, respectively, and therefore the same objections apply also to these claims.

700 IBM Technical Disclosure Bulletin
38(1995) Noember, No. 11, Armonk, NY, US

IBM® Technical Disclosure Bulletin Vol. 38 No. 11 November 1995

Forwarder Dynamic Link Libraries as a Method for Servicing Software

XP 000547408

p. 407-408 = ②

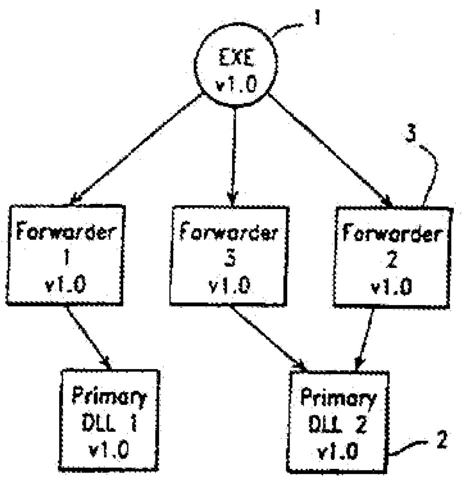


FIG. 1.

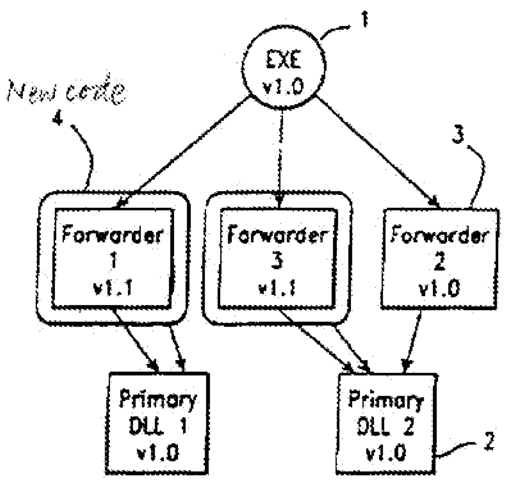


FIG. 2.

Disclosed is a method for servicing and packaging enhancements for a software product, which is shipped with one or more large, primary Dynamic Link Libraries (DLL) and several smaller, secondary forwarder DLLs. Calls to the primary code are made through the forwarders. Software fixes and enhancements are provided by replacing individual forwarder entries with a corrective function inside a secondary DLL.

Most applications, together with modern operating systems, use one or more DLLs to supply commonly-used functions efficiently. A DLL may be used as a forwarder to one or more other DLLs, with code being linked to a forwarder DLL, instead of to the primary DLL, implementing the desired functions. The loader resolves the forwarder by loading the primary DLL and by setting the call address in the calling code directly to the address of the code in the primary DLL.

With the presently disclosed method, software products imple mented using forwarder DLLs may be serviced or enhanced by simply replacing forwarders in the forwarder DLLs with corrective functions. These corrective functions may replace existing functions by reimplementing them without calling the original functions, or they may enhance functions by adding a prologue and/or an epilogue to the call.

Fig. 1 is a block diagram showing an application shipped with an executable 1 (EXE), two primary DLLs 2, and three forwarder DLLs 3.

Fig. 2 is a block diagram showing the same basic components with newer versions of two of the forwarder DLLs. New code 4 has been added to these forwarder DLLs, as depicted by outer boxes. In general, some calls by the application into the secondary DLL are still forwarded, some cause the execution of new corrective code in a forwarder DLL, and some execute new corrective code in addition to being forward.

The use of this method provides advantages if it becomes necessary to ship corrected or enhanced software to customers. Fewer disks are generally required to ship such software, since the size of the forwarder DLLs, even with the corrective functions, is likely to much smaller than that of replacement primary DLLs. Furthermore, with properly designed and documented interfaces, fixes and enhancements in the forwarder DLLs may be written by software developers with no access to the underlying primary code. Even if the primary code is encumbered by royalties and copyrights, the fixes and enhancements need not be so encumbered.

(PCT.195) 700 IBM Technical Disclosure Bulletin
39(1996) January, No. 1, Armonk, NY, US

E

Dynamic Interception of Imported Procedure Calls

XP 000556373 p.197-201 = 5

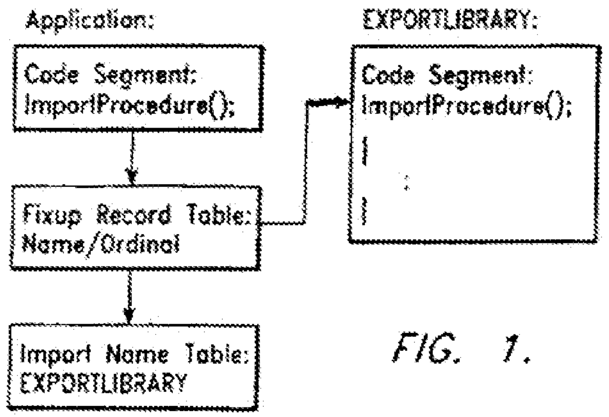

Application:
Code Segment: ImportProcedure();
Fixup Record Table: Name/Ordinal
Import Name Table: EXPORTLIBRARY
EXPORTLIBRARY:
Code Segment: ImportProcedure();
{
:
}

FIG. 1.

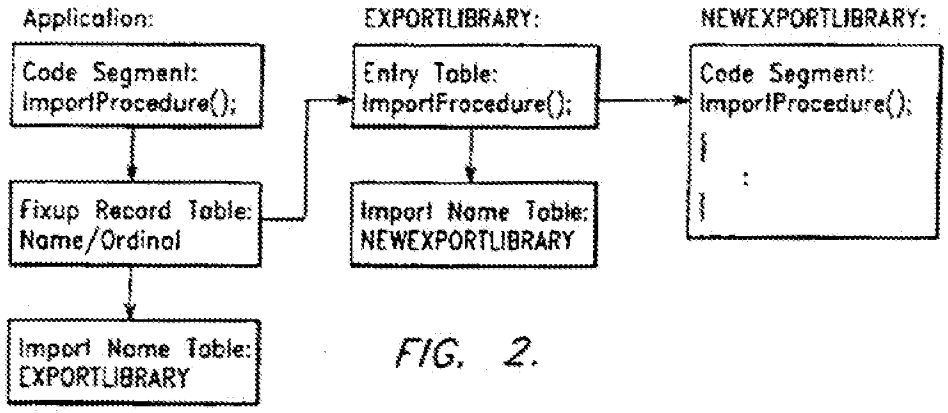
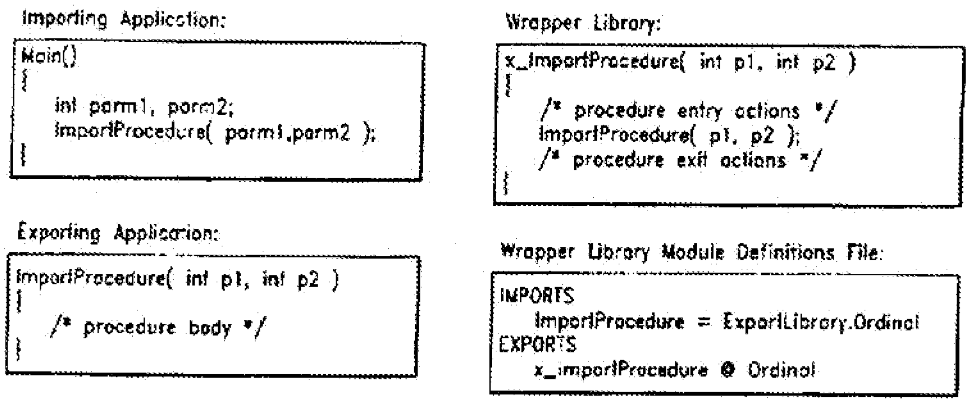

Application:
Code Segment: ImportProcedure();
Fixup Record Table: Name/Ordinal
Import Name Table: EXPORTLIBRARY
EXPORTLIBRARY:
Entry Table: ImportProcedure();
Import Name Table: NEWEXPORTLIBRARY
NEWEXPORTLIBRARY:
Code Segment: ImportProcedure();
{
:
}

FIG. 2.

Disclosed is a method for dynamically intercepting imported procedure calls without requiring alterations to the source code of the importing or exporting applications.

Software organizations have a critical need to analyze the programming interfaces provided by both operating systems and applications. For example, function and system test organizations need to trace procedure input and output parameters, as well as return codes, to determine the code path coverage being attained and to determine whether procedures are functioning properly. Performance organizations are interested in measuring the length of time a procedure requires to accomplish each of its tasks. Customer support needs to determine program logic flow to get a better understanding of why a customer's application fails.

(PCT.195) 700 IBM Technical Disclosure Bulletin
39(1996) January, No. 1, Armonk, NY, US



```
Main()
{
    int parm1, parm2;
    ImportProcedure( parm1,parm2 );
}
```

```
x_ImportProcedure( int p1, int p2 )
{
    /* procedure entry actions */
    ImportProcedure( p1, p2 );
    /* procedure exit actions */
}
```

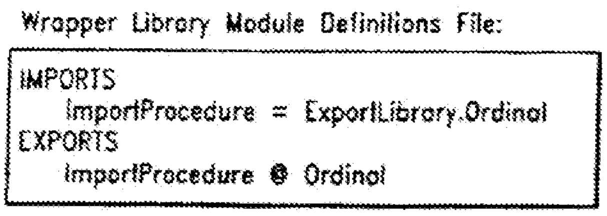
```
ImportProcedure( int p1, int p2 )
{
    /* procedure body */
}
```

```
IMPORTS
    ImportProcedure = ExportLibrary.Ordinal
EXPORTS
    x_ImportProcedure @ Ordinal
```

FIG. 3.

The conventional methods for meeting these needs are either to incorporate procedure wrappers within the application invoking (importing) the programming interface, or alternately to incorporate hooks in the procedures within the application defining (exporting) the programming interface. While these wrappers and hooks can easily trace procedure input and output parameters and return codes, and can easily measure timing by capturing procedure entry and exit times, an underlying problem with each of these methods is that modification is required to source code of the importing or exporting application, along with recompilation, relinking, and redistribution of the finished product. The nature of this problem eliminates the possibility of intercepting and analyzing procedures imported by off-the-shelf applications, as the source code is generally unavailable.

The presently-disclosed method relies on the fact that operating systems supporting dynamic linking require a means for resolving dynamic link references, to successfully load applications which import procedures. This resolution may be achieved by generating a relocation record for each imported procedure that contains the module name and the procedure name or ordinal. Another method, which reduces or optimizes link time, generates a relocation record for each imported procedure that contains the index into a module name table as well as an ordinal or an index into a procedure name table. In this way, only a single module name string per module and a single procedure name string per procedure is required, regardless of the number of invocations. Another method, which reduces or optimizes load time, generates code for each imported procedure, with the code jumping to common code invoking the procedure. This method requires only a single relocation record per procedure, regardless of the number of invocations, with the relocation record using one of the previously-described methods to identify the module and procedure. Combinations or variations of these methods are also possible, but the name of the module exporting the procedure must reside somewhere within the application importing the procedure. A subsequent change of the name causes a new module to be loaded instead of the old module, with corresponding procedures being called in the new module instead of in the old module.


Wrapper Library Module Definitions File:

```
IMPORTS
    ImportProcedure = ExportLibrary.Ordinal
EXPORTS
    ImportProcedure @ Ordinal
```

FIG. 4.

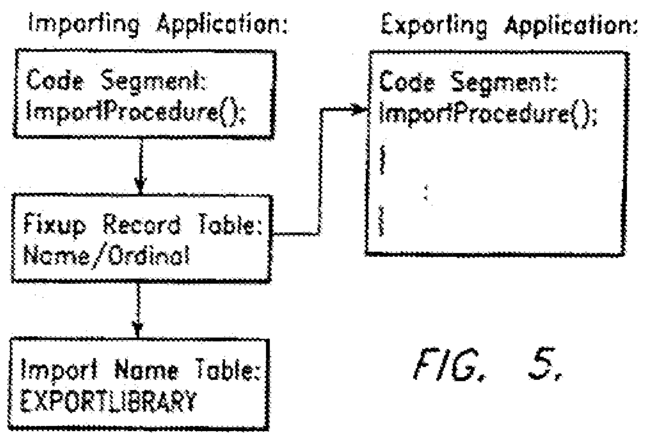


FIG. 5.

Fig. 1 is a block diagram showing how these concepts are applied in the linear executable module format of OS/2*, with the import module name table defining the module name strings imported through dynamic link references. Each imported procedure called by an OS/2 application has a matching relocation record, which references the appropriate string within this table.

The presently-disclosed method also relies on the fact that operating systems that support dynamic linking require a means of defining where an exported procedure resides within a module. Information on where the procedure begins within a particular code segment is usually grouped together in some form of table. Some operating systems have developed means to forward procedures from one module to another, allowing a procedure to be imported from one module even though the code physically resides in a different module. Subsequently, procedures which do not need to be intercepted can be forwarded to the true exporting application by both importing and exporting the procedures. While forwarding is not needed for the presently-disclosed method, it provides an advantage in that only the procedures of interest need to be intercepted. Without a form of forwarding mechanism, every procedure within a module must be intercepted.

Fig. 2 is a block diagram showing how these concepts are applied within the linear executable module format of OS/2, with an entry table containing object and offset information that is used to resolve relocation references to the entry points within a module. The forwarder entry of the entry table is an exported entry point that resolves to an imported reference. Thus, the module contains no code associated with the forwarded entry point, and the final target address of the forwarded entry is contained in another module.

Fig. 3 is a block diagram illustrating the intercepting and analyzing of imported procedures by the presently-disclosed method. Procedure wrappers are incorporated into the source code of the wrapper library, which is a Dynamic Link Library (DLL) having an entry table containing an entry for each entry in the entry table of the exporting application's DLL. The wrapper library contains a procedure wrapper for each procedure of interest in the exporting application. The wrapper library intercepts these procedures by placing the names of the wrapper procedures into the wrapper library's module definitions file.

Fig. 4 is a block diagram illustrating how the presently disclosed method passes on interrupted procedures without intercepting them. The wrapper library does not contain a procedure wrapper for any procedure not of interest in the exporting application. Such a procedure is forwarded by placing the procedure name in both the imports and exports section of the wrapper library's module definitions file. Once the wrapper library is available, the imported procedures can be intercepted and analyzed by simply replacing the name of the exporting library within the import module name table of the importing application with the wrapper library's name.

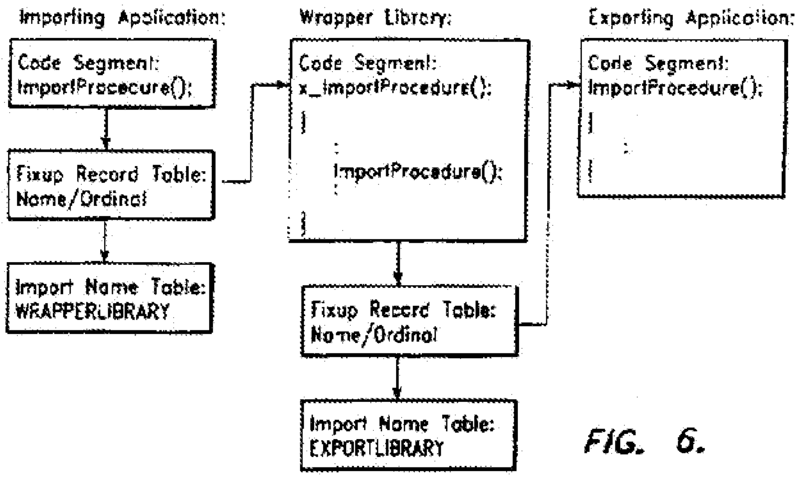

Importing Application:
Code Segment: ImportProcecure();
Fixup Record Table: Name/Ordinal
Import Name Table: WRAPPERLIBRARY
Wrapper Library:
Code Segment: x_ImportProcedure();
{
ImportProcedure();
}
Fixup Record Table: Name/Ordinal
Import Name Table: EXPORTLIBRARY
Exporting Application:
Code Segment: ImportProcedure();
{
}
FIG. 6.

Fig. 5 is a block diagram showing the interface between importing and exporting applications before the import module name table is modified.

Fig. 6 shows the interface of Fig. 5 after the import module name table is modified. If the name of the wrapper library is the same length as the name of the exporting library it is replacing, the name need only be overwritten in the import module name table of the importing application. Otherwise, the entire import module name table, and everything in the importing application following the table, need to be adjusted and rewritten, along with the appropriate entries in the linear executable header.

With the wrapper library available, having its name in the import module name table of the importing application, all procedures imported by the application are redirected first through the

(PCT.195) 700 IBM Technical Disclosure Bulletin
39(1996) January, No. 1, Armonk, NY, US

wrapper library. If a procedure wrapper exists, control is passed to the wrapper, then to the exporting application, and then back through the wrapper. If a procedure wrapper does not exist, control is passed directly to the exporting application through a forwarder. As a result, when wrapper insertions, modifications, or deletions are required, only the wrapper library needs to be recompiled, relinked, and distributed; both the importing and exporting applications remain unchanged.

To discontinue intercepting the procedures, the wrapper library's name within the import module name table of the importing application is changed back to the name of the exporting application.

* Trademark of IBM Corp.